MULTIMEDIA UNIVERSITY

STUDENT ID NO

					1	5	1	0	1

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 1, 2019/2020

ECE3136 – JAVA TECHNOLOGY

(All Sections / Groups)

19 OCTOBER 2019
2:30 P.M – 4:30 P.M.
(2 Hours)

INSTRUCTIONS TO STUDENT

1. This examination paper consists of SEVEN (7) pages including cover with THREE (3) questions only.
2. Answer ALL questions. All questions carry 25 marks and the distribution of the marks for each question is given.
3. This examination is OPEN-BOOK. Allowed material: Textbook and/or bound notes only.
4. Please print all your answers in the Answer Booklet provided.

Question 1

Figure Q1a shows the code of a class, `Dimensions`.

```
public class Dimensions
{
   private double width;
   private double length;

   public Dimensions (double w, double l)
   {
      setWidth (w);
      setLength (l);
   }

   public void setWidth (double w) { width = w; }

   public double getWidth () {      return width;     }

   public void setLength (double l) { length = l;     }

   public double getLength () { return length;  }

   public String toString () { return width + "\tLength: " + length;
   }
}
```

Figure Q1a: Class `Dimensions`

Write a Java application class, `AreaCalculator`, which calculates the area of a house. It does this by prompting the user for the width and length of the rooms in the house. Use an `ArrayList` of `Dimensions` objects to store the room dimensions entered. Allow the user to enter multiple room dimensions repeatedly. See Figure Q1b.

[12 marks]

After user enters the input 'n', indicating no more rooms to input, compute and display the list of rooms numbered from 1 onwards, their dimensions, and total house area. Format the calculated area to be 2 decimal places. Display the total house area in a dialog box (see Figure Q1c). For houses which are more than 1264 sqft, display a second dialog box to tell the user that their house is bigger than average. (see Figure Q1d).

[11 marks]

Handle all exceptions that may occur.

[2 marks]

Continued...

```
Enter Width (in feet): 78
Enter Length (in feet): w
Invalid input
Another? (y/n): y
Enter Width (in feet): 39
Enter Length (in feet): 30
Another? (y/n): y
Enter Width (in feet): 20
Enter Length (in feet): 20
Another? (y/n): y
Enter Width (in feet): w
Invalid input
Another? (y/n): n
Room 1: Width : 39.0 ft x Length : 30.0 ft
Room 2: Width : 20.0 ft x Length : 20.0 ft
```

Figure Q1(b)

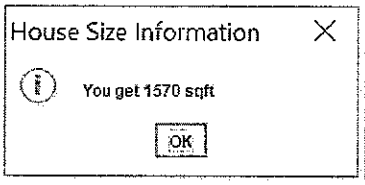

Figure Q1 (c)

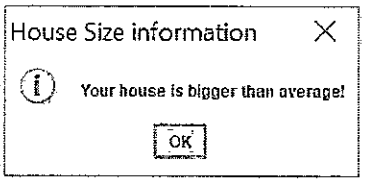

Figure Q1 (d)

Continued...

Question 2

Create a Java application that can compute the future value of a fixed deposit, given the principal amount, number of years and interest rate, and highlights when the depositor's principal will be doubled. The future value formula:

$$Future\ Value = P \times (1 + \tfrac{r}{100}/n)^{Yn}$$, where

P is the principal amount,
r is the interest rate per annum in percent (Eg. For interest of 4%, $r = 4$),
n is the number of times of compounding per year,
Y is the number of years

(a) Write a class, **`CompoundInterestPanel`**, which is a **`JPanel`** class. Apart from the GUI components, this class has one type **`int`** instance variable: **`compoundingTimes`** and one type **`double`** instance variable: **`interest`**. The constructor of the class accepts two arguments, **`ct`** (type **`int`**) and **`rate`** (type **`double`**). These arguments are used to initialize **`compoundingTimes`**, which is the number of times the interest is compounded within a year, and **`interest`**, which is the annual interest rate (in %). The panel contains GUI elements as shown in Figure Q1a. One label displays the number of compounding times and annual interest rate, and two more labels describe the input required at each of the two text fields. After the button "Calculate", include a label to display output after the computation of future value. By default, the output label displays an empty string.

[11 marks]

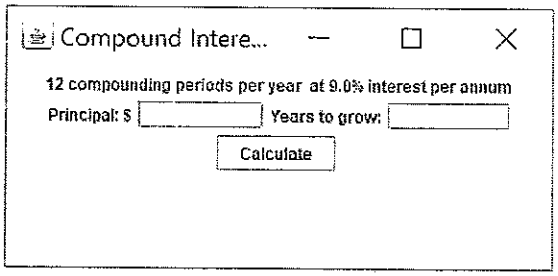

Figure Q1a

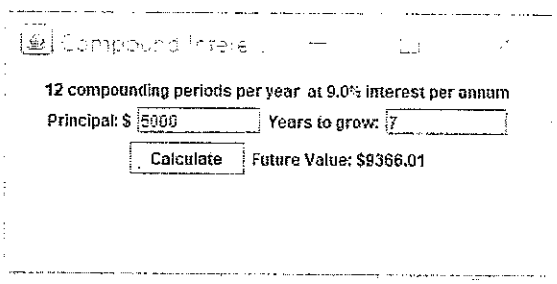

Figure Q1b

When the button "Calculate" is clicked, obtain the inputs entered in the text fields and perform the computation to obtain the future value amount using the input values along with variables **`compoundingTimes`** and **`interest`**. Display the result using the output label. See Figure Q1b. Subsequently, evaluate the user's interest rate. If the interest rate is equal to more than 10%, highlight that the user's money will double is 7 years or less (See Figure Q1c). If the interest rate is less than 10%, highlight that user's money will double in the number of years according to the Rule of 72 (See Figure Q1d).

Rule of 72 says:

Number of years to double your principal = $\frac{72}{r}$, where

Continued...

r is the interest rate per annum (in percent)

[10 marks]

Handle all exceptions that may occur.

[2 marks]

(b) An application class intends to create an instance of **`CompoundInterestPanel`** for a compounding 6 times a year and 7.5% interest per year. Write the **object instantiation statement**. (*Note*: **DO NOT write the entire application class**)

[2 marks]

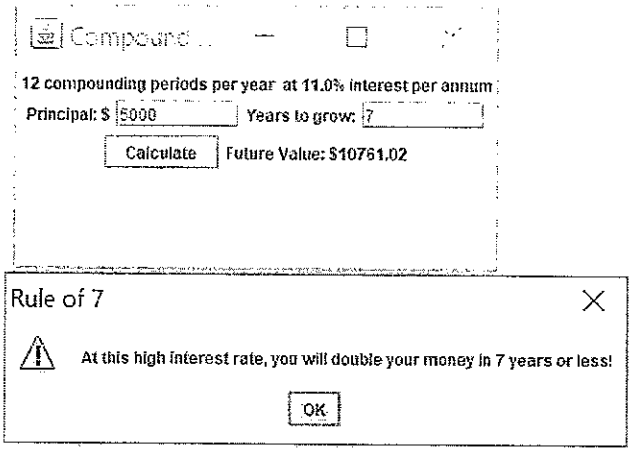
Compound...
12 compounding periods per year at 11.0% interest per annum
Principal: $ 5000
Years to grow: 7
Calculate
Future Value: $10761.02
Rule of 7
At this high interest rate, you will double your money in 7 years or less!
OK

Figure Q1c

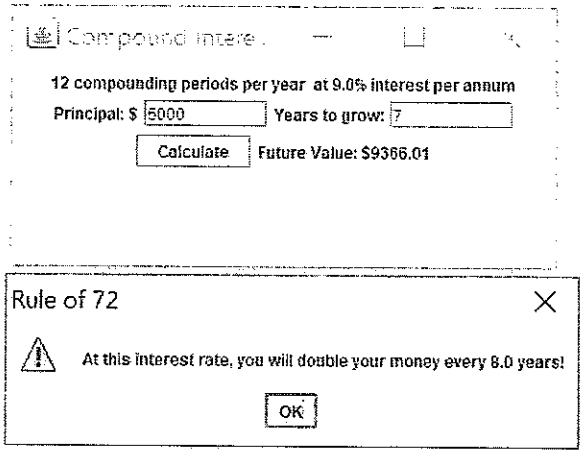
Compound Intere...
12 compounding periods per year at 9.0% interest per annum
Principal: $ 5000
Years to grow: 7
Calculate
Future Value: $9366.01
Rule of 72
At this interest rate, you will double your money every 8.0 years!
OK

Figure Q1d

Continued...

Question 3

(a) Write an abstract class, **`Account`**. Include the attributes **`accountNo`**, **`principal`** and **`interestRate`**, where **`accountNo`** identifies the account, **`principal`** is the amount invested, and **`interestRate`** is the interest rate per annum. Provide a parameterized constructor which calls the mutator methods. Provide all necessary accessor and mutator methods and a **`toString`** method. Provide an abstract method **`calculateInterest()`** which returns a **`String`**.

[7 marks]

(b) Write a class, **`FixedDeposit`**, which inherits from **`Account`**. Include attribute **`tenure`**, where tenure is the number of months for the investment. Provide a constructor, and all necessary accessor and mutator methods. Override superclass abstract method **`calculateInterest`** to return a summary to show the fixed deposit account's total interest earned at the end of the tenure. Format the interest earned value to be exactly 2 decimal places. Do not override the superclass **`toString`** method.

*Fixed Deposit Interest earned = principal * interestRate/12 * tenure*

[7 marks]

(c) Write another class, **`Savings`**, which inherits from **`Account`**. It has no other attributes other than those inherited. Provide a constructor. Override superclass abstract method **`calculateInterest`** to return a summary to show the savings account's total interest earned per annum. Do not override the superclass **`toString`** method.

*Savings Interest earned = principal * interestRate*

[4.5 marks]

(d) Write a Java application class, **`Accounts`**, which displays the accounts with their total interest earned and current principal. Use an **`ArrayList`** of **`Account`** objects, **`acc`**, to store the accounts. Create an account of type **`FixedDeposit`** and an account of type **`Savings`** as test cases and store them into **`acc`**. Using **`acc`**, display the list of accounts together with their respective interest earned (use polymorphism). See Figure Q3a.

[6.5 marks]

Continued...

```
Accounts:
Account Number: FD123130
Principal: 5000.0
Interest rate: 0.05 per annum,
Interest for 9 months: RM187.50

Account Number: SA123131
Principal: 5000.0
Interest rate: 0.0035 per annum,
Interest for one year: RM17.50
```

Figure Q3a: Sample output, where the first account is Fixed Deposit and the second account is Savings

End of Paper